PROGRAM PROTECTION METHODS FOR THE ATARI *

An educational document

By James J. Justin

Copyright (c) 1983

MANDALA MEDIA

*ATARI is a trademark of ATARI INC.

TABLE OF CONTENTS

PAGE

INTRODUCTION 2

THE AUTORUN.SYS 5

LOCK OUT THE BREAK KEY 6

MACHINE LANGUAGE BOILERPLATE.... 7

BAD SECTORING 9

MACHINE LANGUAGE INSIDE BASIC .. 11

CLOBBER THE VARIABLES 13

CHOP OFF THE END 15

MOVING THE DIRECTORY 17

ENCRYPTION 19

MAKE THE PROGRAM HARD TO READ .. 21

MAKE THE PROGRAM HARD TO MODIFY. 23

COVER YOUR TRACKS 25

GET PROFESSIONAL HELP 27

LEGAL PROTECTION 28

CONCLUSION 31

REFERENCES 32

NOTICE

Copyright (c) 1983. All rights reserved. This document may not in part, or in whole, be copied in any format without the prior written consent of Mandala Media.

DISCLAIMER

Mandala Media assumes no liability or responsibility to the purchaser or any other entity or person with respect to any damage or loss, or alleged damage or loss caused directly or indirectly by the ideas and techniques presented in this document. This disclaimer includes but is not limited to any loss of business, any loss of anticipated profits, any interruption in business or any other damages resulting from or allegedly resulting from the use of ideas presented in all or any part of this document. If your State does not allow any of the statements or representations contained in this disclaimer, then those shall not apply to you. You may also have other rights which depend upon your State of residence.

All information presented in this document is presented in good faith and is accurate to the best of our knowledge. Responsibility for detecting possible errors, omissions or any other discrepancies is with the reader. The use of information presented in this document is at the readers own discretion and risk. Neither the Author nor Mandala Media assumes any responsibility whatsoever for the use of these ideas or techinques in the products of others.

Mandala Media,

P.o.Box 7082,

San Diego, Ca, 92107

INTRODUCTION

This is the first edition of "PROGRAM PROTECTION METHODS FOR THE ATARI". We know that we are merely scratching the surface of the body of all ATARI software protection methods that are possible, but it is a start, and it is hoped that this edition will be followed by many more. It is now up to you, the programmers and authors out there to provide some new ideas that will help to protect us from unauthorized copying of our programs. If you have any ideas at all, tested or untested that you think may supplement or enhance the techniques outlined in this document, please send them in. If any of your ideas are used, you will be given credit in our next edition of "PROGRAM PROTECTION METHODS FOR THE ATARI". Of course, if you prefer to remain anonymous, your wishes will be respected. In any case, anyone sending in an idea that is accepted for inclusion will receive a free copy of the next edition of "PROGRAM PROTECTION METHODS FOR THE ATARI".

Yes, it is possible that some of those who would copy our programs may also be reading this document, but the methods outlined should still make things difficult, because knowing how to implement a technique and undoing it are usually two very different affairs. Most of these techniques are a lot more difficult to undo than to implement.

There are essentially two ways in which programs may be copied. The first is by direct copy. This method produces an exact copy of the original for use by a non purchaser of the program. The second is not by direct copy, but is through the use of the logic and techniques contained in your program.

This second copy mode does not produce an exact copy, but a slightly different program that is similar in logic and technique. This document presents methods that may be helpful in thwarting either or both of these copy modes.

Though many of the concepts presented here may also apply to protection of tape oriented programs, "PROGRAM PROTECTION METHODS" is essectially disk oriented in nature.

We would like to mention here in the introduction that to our knowledge, there is as yet no foolproof way to protect your program from copying. No matter what kind of protection you implement, someone will come along who can break it, but the point is that with the ideas presented here, you can continue to make things increasingly difficult to the point where attempting to copy your program will simply not be worth the effort.

On the other side of the coin, there is currently a controversy brewing as to just how much protection should be put into a program. When the protection starts making a program difficult to use or error prone, then you have reached a point of diminishing returns. At that point, people will stop buying your programs. There lies a very thin line between what is enough and what may be too much; only you can decide where that line is.

Since many of the sections in this document are interrelated, it might be a good idea to quickly skim through the all of the sections once, then go back through, reading more carefully for the details.

Last, we must state that due to the complexity of the subject, it is assumed that readers are knowledgeable regarding programming terms and are familiar with the ATARI computer. We are assuming that if we present some concepts and give you some magazine, book, and manual references, that you should then be able to work out the details for yourself. Some of the techniques discussed will apply to machine language programs, some will apply to BASIC and some will apply to both. The reader should be able to tell the difference by the context in which the technique is described.

Ok, enough for the introduction, lets get down to the nitty gritty of how you can protect your programs.

THE AUTORUN.SYS

This section works in connection with the "LOCK OUT THE BREAK KEY" section that follows. As will be mentioned here and again, one way to keep pirating eyes out of your program is to seize program control at the very moment the computer is turned on and keep control inside your program until the computer is turned off.

You may accomplish this through the use of an AUTORUN.SYS program file. An AUTORUN.SYS program file is a special file recognized by the ATARI Disk Operating System (DOS). If a file named AUTORUN.SYS exists on a currently inserted diskette, that program file will be automatically loaded when the computer is first turned on. When an execution address is specified in the AUTORUN.SYS file, program control will be immediately passed to that address causing the AUTORUN.SYS program to run, and thus take control of the computer.

This particular kind of AUTORUN.SYS file should be an initializing machine language program that will load and execute the MAIN BASIC or MACHINE LANGUAGE program. If the MAIN program is designed to lock out the break key and system reset key, (see "LOCK OUT THE BREAK KEY ") then the entire operation of your program may be accomplished in a locked out (to the user) mode.

Note: MANDALA MEDIA has a program that will generate an appropriate AUTORUN.SYS program/file automatically for you. This program will be very reasonably priced. Please let us know if you are interested.

LOCK OUT THE BREAK KEY

As long as your program is able to keep control of the computer, no one will be able to stop it to look at it or list it. There are three things you should do to help your program maintain that control:

1)Lockout the break key. This can be accomplished by putting a couple of poke* commands into a subroutine of your program as follows:

```
100 POKE 16,64: POKE 53774,64:RETURN
```

This subroutine should be called periodically from your main program. There are some BASIC commands that will re-enable the break key and therefore your subroutine should also be called following each of those types of operations. These BASIC commands are PRINTs or OPENs to the screen (S: or E:) or PRINTs after screen OPENs and also GRAPHICS commands.

2)Cause the system to reboot (cold start) whenever the system reset key is pressed. This is done with another poke* probably (but not necessarily) placed in the same subroutine as the break key lockout pokes above. The total subroutine would now read:

```
100 POKE 16,64: POKE 53774,64: POKE 580,1: RETURN
```

3)Make sure all potential program errors, disk errors and end of file indications, etc, are TRAPPED and handled by the program internally (See discussion of trap in Basic Reference Manual page 22). In other words, never allow the program to stop for any reason, as that would be relinquishing control and would give the pirate a wedge into your program.

* Or the machine language equivalent if your program is machine language.

MACHINE LANGUAGE BOILERPLATE

A BASIC or machine language program can be protected by the use of a direct machine language boot file program on disk (See ATARI OPERATING SYSTEM USERS MANUAL Pages 18, 142-149 & Section 7). The ATARI system when turned on with a ready to load disk connected, will load up to 256 sectors then transfer control to the machine language program that was loaded. Once executed, the executed program can load additional sectors from disk and in turn, transfer control to those program instructions. This is another method of seizing control of the computer at turn on time and not letting it go until the computer is turned off. Of course, your running program must also lock out the break and system reset keys, and handle program errors, disk errors, and end of file indications appropriately so that control is maintained within your program at all times. (see "LOCK OUT THE BREAK KEY"

This boot file program or your main program can also check to see if the disk currently loaded is an original or just a copy of an original. For the details on this, see the section on "BAD SECTORING".

This method of loading and executing a program from a boot file is usually used in loading and executing machine language programs, but it can be used with BASIC programs as well. The only problem with loading BASIC programs by this method (and it is significant) is that you must make everything look to BASIC like it would have looked if BASIC itself had loaded the program. This means setting up all of the pointers etc. that BASIC will need before your boot load program issues a "run" command to BASIC (See chapter 10 of DE RE ATARI listed in the REFERENCES section.

An added benefit of loading your programs by this method is that it is possible to do decryption while loading a program residing on disk in an encrypted form. (see section on "ENCRYPTION ")

NOTE: If there is enough demand, we are considering the development of a package containing a boot file program with all of the machine language instructions needed to implement this method. The proce for this package will be reasonable. If you think you might be interested, please let us know.

BAD SECTORING

Believe it or not, one or more BAD sectors on a disk can help to copy protect a program on that disk. The theory on this method is as follows:

At the time you create each disk you intend to distribute, containing the programs or programs you want to protect, you locate one or more unused sectors on the disk and crash them. To crash a sector means to slow your disk drive way down to the lowest level at which your disk will still operate (200-220 rpm) and then write any data to the sector you want crashed. This implies that you must physically open your disk drive to adjust the speed regulator inside the disk. (Keep in mind that this process will invalidate any warantee you may have on the disk). The effect of all this is that those crashed sectors become unreadable when a disk drive is running at its normal speed of 286-290 rpm.

You will need a disk speed check program (many of which are available in the market place) to set your disk to the slow speed and back to normal again. These programs usually come with directions on opening your disk and adjusting the speed.

In addition to crashing some sectors on the disk, you must also have changed your program to check on (read) those sectors that have been crashed.

Now, when your purchased program gets control at start up time, you have programmed it so that it attempts to read the known BAD sector(s). If the program IS UNABLE to read the sector(s), then everything is fine and the program is allowed to function normally.

On the other hand, if your program IS ABLE to read those sector(s), then the disk must be a copy of the original, and you have designed the program to crash or do something bad, like reformat the disk.

Why does the ability to read those BAD sectors on the disk mean that the disk is a copy ??? Simply because most copy programs do not copy bad sectors from an original disk as mirrored bad sectors on the new (copy) disk. Copy programs usually just skip bad sectors leaving blank (but good) sectors where there were bad sectors on the original. When it is detected that sectors that were originally bad, are no longer bad, then this must surely be a copied

MACHINE LANGUAGE INSIDE BASIC

Using BASIC strings that contain machine language programs in conjunction with the BASIC USR call, you can implement some protection for BASIC programs that will make your program on disk difficulty to copy and difficult to list or read (For discussion of the BASIC USR call, see the Basic Reference Manual & the Assembler Editor Manual)

A machine language string inside BASIC can make your program difficult to copy by use of the method described in the section on "BAD SECTORING". When your machine language program inside a string is USR called by your BASIC program, it should check a sector or sectors (that have been intentionally crashed on your original disk) to see if they are still bad. If the current disk is a copy of the original, then any sectors that were originally bad, will now be good sectors containing no data (most copy programs merely bypass bad sectors). If a disk sector that was originally bad is now able to be read, your machine language program in the BASIC string has been designed to go into a continuous loop, or do a cold start, or even reformat the disk (If you feel a little vindictive towards the person trying to "RUN a COPY of your program).

A machine language string within BASIC will also make your program difficult to read in several ways. First, machine language is just inherently more difficult to read than is most BASIC code. Second, machine language contained in a string doesn't look like assembly language or even machine hex code. It usually generates some weird looking ATASCII characters when listed to the screen, and will cause printers to do all kinds of strange things because some of the machine language numbers are interpreted by a printer to be printer control codes.

Sometimes a printer will just give up completely and stop printing when trying to print a machine language string inside a BASIC program.

NOTE: We are currently developing a program to help you with the creation of BASIC strings containing machine language. This program will allow you to enter machine language codes and will produce appropriate BASIC strings that may be "listed" to disk and then "entered" into any BASIC program where you may need the machine language program contained in that string. The price for this program should be very reasonable. If you are interested in obtaining a copy, please let us know so we can send you further information as soon as it becomes available.

CLOBBER THE VARIABLES

By clobbering the variables, it is possible to make your BASIC program unlistable or so garbled that it is not readable. This may be accomplished by changing each of the variable names from its normal ALPHA-NUMERIC ATASCII characters to NON ALPHA-NUMERIC (unlistable) ATASCII characters. To do this, add the following BASIC lines to the end of your program:

```
32760 VRBLNAMESTART = 256 * PEEK(131)+PEEK(130)
32761 VRBLNAMEND = 256 * PEEK(133) + PEEK(132)
32762 FOR VRBLADDRESS = VRBLNAMESTART TO VRBLNAMEND
32763 POKE VRBLADDRESS,0: REM Make vrbl unlistable
32764 NEXT VRBLADDRESS
```

Actually, these statements could all be put on one line. They are spread out here to make the statements easier to read and understand. Also, any other unlistable ATASCII character could be used in place of the zero in line 32763.

Now just before you are ready to save your program with the SAVE "D:FILENAME" command, type GOTO 32760 to execute these statements, ten try to list your program. The program should either be unreadable, or depending on whether there were more than 256 characters in the variable name table, will just scroll garbage characters continuously. It may not list at all. If you type RUN however, everything works normally.

Now, if it is still scrolling, stop the listing by pressing the system reset key, then while your program is still in this clobbered condition, do the SAVE to disk mentioned in the previous paragraph. Though it will still RUN perfectly, the program will remain unlistable even after it has been saved and reloaded. (Try it)

At first glance, this would appear to be a foolproof way to make your program permanently unlistable, but don't be too sure. It is possible to reverse the process by putting listable characters back into the variable name table. To be sure, the original variable names have been lost, but the program does become listable again with a new set of variable names.

So much for clobbering the variable. This technique is not an end all in protecting your program, just one more thing you can do to make things difficult.

For background information on BASIC, see DE RE ATARI, the chapter on BASIC or The ATARI Tutorial on BASIC, Feb. 1982 BYTE Publications Inc.

CHOP OFF THE END

The method described in this section will make a saved BASIC program totally unlistable.* In fact, after using this technique, the only way you will be able to use your program at all, will be with the RUN "D:FILENAME" command. If you try to run the program in two steps by typing LOAD "D:FILENAME" followed by typing RUN, the program will not execute.

This could work out very nicely with the use of an AUTORUN.SYS file in combination with a BASIC program, because the AUTORUN program can be made to issue a RUN "D:FILENAME" command as its last step in giving control over to BASIC. This overall process would be in keeping with the concept of holding the internals of the program locked out to the user from the start of execution to termination.

The technique is implemented as follows:

At the very end of your program on the last line, type in the line:

```
32767 CURRENTSTMTADR=256 * PEEK(139) + PEEK(138): POKE
CURRENTSTMTADR+2,0: SAVE "D:YOURFILENAME"
```

Now, with a blank formatted disk in your disk drive, type (from the direct mode) GOTO 32767.

*Totally unlistable here means unlistable by the BASIC cartridge "LIST" command. Don't think that this is an endall method of preventing the listing of your program. This again, is just one more way of making things a little more difficult.

Once you have learned that the first byte following the line number (CURRENTSTMTADR + 2) in each tokenized BASIC line is reserved for the total line BYTE COUNT, it becomes a little easier to understand what will be happening as line 32767 is executed.

The number of bytes in the current BASIC line (held at CURRENTSTMTADR+2) is set to zero causing the BASIC "direct mode" * line (next line at 32768) to be chopped off. If there are zero bytes in the current line, then you can never get to the next line. The SAVE "D:YOURFILENAME" in line 32767 automatically saves your altered program on disk.

Now, when your program is loaded back in, it loads ok, but without any "direct mode" BASIC line at 32768. Since there is now no "direct mode" line in which to process "direct mode" BASIC commands, it becomes impossible to issure any "direct mode" statements to BASIC and you will not be able to LIST or RUN your program. Try it!

Now, turn your computer off, then turn it back on again to get back to a refreshed BASIC. If you issue a RUN "D:YOURFILENAME" command now while the "direct mode" BASIC line is still intact, the RUN part of the command will have already been accepted by BASIC before your program is loaded and everything will work fine. A NEAT TRICK!

* The BASIC "direct mode" line is an invisible line always held by BASIC as line number 32768.

MOVING THE DIRECTORY

Another way of denying easy access to your BASIC programs is to move the disk directory to a new non standard location on your disk, then whenever an attempt is made to display the directory, a potential pirate will see nothing or maybe a canned message that you have placed (with the use of a disk editor program) in the normal directory location on disk.

Once you have moved the directory by this method, the program names cannot be seen (with normal methods) nor can the program be loaded by the standard BASIC loading commands even if you do happen to know the program name. A normally oriented system does not even know that your program exists.

All it takes to implement this technique is the following:

Before saving your program, do a "direct mode" POKE from BASIC.

```
POKE 4226,112 (RETURN)
```

After you have poked the memory location at 4226 with 112 (normally, 4226 contains 105), you should immediately save your program to disk by the usual SAVE "D:FILENAME" command. Now turn off your computer, then reboot the system, go to DOS and try to display the directory. Zap, your program is not there.

Next, use the "B" option to get back to BASIC, then try to load your program using the LOAD "D:FILENAME" command. You should get nothing but an error 170 indicating your file is not on the disk. But you just saved it ??? Is it there or not? YES, it <u>is</u> on the disk, but because of your poke before the save, the directory has been moved.

To prove that this is the case, just repeat the same poke to 4226 that you did before (POKE 4226,112), then try to LOAD "D:FILENAME" again. Presto, your program now loads like a charm.

The thing you must remember in all this is that you must reverse the process to load from your displaced directory. This could easily be done from your AUTORUN.SYS program at startup time. To implement this in your AUTORUN.SYS file, just have it do that same POKE again just before it does a RUN "D:FILENAME" to initiate your main program.

ENCRYPTION

Encryption is the process of changing alphabetic or numeric data into a secret form that is not readable or useful until the process is reversed by what is called decryption. This process is one of the most useful tools of the espionage/counter espionage scene. You know, SPY STUFF !

Encryption of your programs on disk can be just one more means of making your programs a little more (or maybe a lot more) difficult to pirate. Searching through an encrypted disk for particular instructions or patterns can become an almost impossible task for a pirate unless he can first decrypt the disk, and that can be a very time consuming process.

There are many methods of encryption, from something so simple as merely adding some constant value (say +5) to each byte (so it will merely be more difficult to read) to bit flipping within each byte which can be accomplished with the 6502 machine language "EOR" instruction. Of course, there are also some extremely complex ciphering methods, too (see Info World * April 18, 1983 pages 27-35 and Creative Computing Magazine **, May, 1983 pages 166-195).

The point to remember here is that whatever encrypting you do to your programs and data as they are stored on disk, must be reversed as they are loaded from disk. This implies that the decryption process must be a part of your loader program and therefore ultimately available to some pirate. (That is if he knows how to find and analyze it.)

* Info World is a publication of Popular Computing, Inc.

** Creative Computing is a publication of Ahl Computing Inc.

MAKE THE PROGRAM HARD TO READ

Using the techniques described in the "CLOBBER THE VARIABLES" and "CHOP OFF THE END" sections, it is possible to make your program at least temporarily "unlistable". These methods will keep most people out of a BASIC program. However, it will stillbe possible to obtain a listing of your program even though it has been made so called "unlistable". If the variables have been clobbered, new variable names can be created, etc, to somehow get a new listing. This listing will not be exactly like the original (the variable names will be different), but it will show the original logic and all of the BASIC commands used in your program.

For this reason, you should possibly not stop at just making your program "unlistable"/ Maybe you should even go against good programming principles (at least in your final sale version) and do everything possible to make the final version of your program difficult to read. This may be done by using some or all of the following techniques:

1) Put data and messages into encrypted BASIC strings, and have your program decrypt them only as they are used.

2) For an added confustion factor, use XIO statements in place of other BASIC statements such as GRAPHICS, OPEN, and CLOSE. (See COMPUTE! * MAGAZINE APRIL 1983 Pages 232-234 for some tricks using the concept.)

* COMPUTE! is published by Small System Services, Inc.

3) Create BASIC variables for the constant values that will never be changed again after being initialized for use in your program. Then, after your variable names have been clobbered as described in the CLOBBER THE VARIABLES sections, your constants as well as your variables will not be readable. This can be done only to the extent that you have not used up the 128 maximum number of variables available in BASIC.

4) Use the BASIC "COM" command in place of the "DIM" command as it does exactly the same thing. Since "COM" is less commonly used than "DIM" many people may not know what it means.

5) Put in lots of GOTO's especially of the type where the goto address is a variable or better yet an algebraic expression.

6) Etc.; the number of these kinds of things you can do to your program make it hard to read is limited only by your imagination and your knowledge of BASIC.

MAKE THE PROGRAM HARD TO MODIFY

Even after implementing your optimum combination of progection schemes designed to keep pirates out of your program, it is best to assume that somewhere, somehow, someone will finally be able to extract your program in a useable and/or listable form. Because of thie possibility, and to make things difficult to the very end, you should consider the possible implementation of a few techniques that are designed to make even a listable version of your program difficult to understand or modify.

This section applies mainly to BASIC programs, as machine language programs in hexadecimal or even disassembled form are by their very nature, already difficult to understand and modify. In the case of machine language, maybe just not structuring the program too highly (though this goes against good programming proctice) will be sufficient.

In the case of BASIC, however, there are several things you can do to give a potential pirate headaches when he tries to analyse and/or modify your program:

1) Just before final publication, run your program through a "MASHER" program to remove all remarks and pack the program statements as tightly as possible (This has the added side benefit of saving a considerable amount of RAM memory). A masher program may also convert many of the most used GOSUB & GOTO locations from constant line numbers to BASIC Variables making your BASIC code more difficult to follow.

2) Renumber your mashed program so that the lines are numbered consectuively with no space for the addition of new lines between. (Yes, the pirate could renumber the program, but with additional changes such as those described in (3) next page, renumbering would create other problems)

Change ON GOSUB or ON GOTO commands to computed GOSUB or GOTO commands where possible. For example the BASIC lines:

```
10 ON VARIABLENAME GOTO 12,13,14

11 GOTO 15

12 A = B: GOTO 15

13 A = C: GOTO 15

14 A = D

15 REM: continue program here.
```

Could be changed to:

```
10 IF VARIABLENAME >0 AND VARIABLENAME<4 THEN GOTO 11 + VARIABLENAME

11 GOTO 15

12 A = B: GOTO 15

13 A = C: GOTO 15

14 A = D

15 REM :continue program here.
```

Now, if the program is renumbered, the program will no longer RUN correctly without additional modifictations. TRY IT!

4 The number of these kinds of things you can do to your program to make things difficult is again only limited by your imagination.

COVER YOUR TRACKS

Once you have implemented a protection technique into your program, you can make things even more difficult for the would be pirate if you can make your protection code hard to find for observation and/or modification. One of the ways to do this is by encryption of all or part of the program (see section on ENCRYPTION)) However, there are also some other methods of concealment that might be of interest here.

If your program is in BASIC, one method is to hide your protection code inside a BASIC string containing a machine language program that does the protection. Your BASIC program can then call the machine language protection program with the BASIC USR command. In this case, it is a good idea to include some other functions (needed by your program) within the machine language string, so that your program will not function if the USR call is removed.

There are many ways to conceal protection within a machine language program. One of the best is to design your program so that the protection instructions are created by your program itself, only after the program has already been executed. To implement this, you need to have an area of your program that is "reserved" so to speak, for this purpose. The "reserved" area can initially contain any innocent looking instructions that you desire. Once your program begins executing, it creates the instructions that will do the protection and stores the into the "reserved" area, which is then executed to do the protection you desire (such as checking for a bad sector on disk, etc.).

Using this kind of technique makes things extremely difficult for a pirate who is trying to search on your disk for certain types of protection instructions, as they will not even exist until after your program has started executing. All kinds of tricky techniques can be used in creating the instructions that will be stored in your "reserved" area. For example, numbers or other machine language instructions could be added together, subtracted, EOR'ed or otherwise manipulated to create the machine language instructions you desire, then stored into the appropriate "reserve" locations for subsequent execution. The ways of making your program create protection instructions itself while executing are limited only by your imagination and your knowledge of machine language.

GET PROFESSIONAL HELP

There are at least two kinds of professional help you should consider to aid you in protecting your software:

1) If you are considering any form of legal protection, (see section on LEGAL PROTECTION) then you should definately consider the services of an attorney.

2 Another strong possibility to consider in the protection of your software is the use of a professional disk duplication service. Some professional diskette copy companies with the use of advances diskette duplicating machines are able to copy protect your software by producing abnormal sector addresses during formatting of the diskettes.

For example, two adjacent sectors in a disk track can be given the same identical sector address completely eliminating another sector address that would normally have been written on the disk. This situation cannot be duplicated by the standard diskette copy programs, and therefore allows your program to check for a selected predetermined situation at initialization time. If your program determines that the diskette is a copy, it merely takes a dive and another potential pirate is thwarted.

(See article on "Disk Backup Programs" page 72, A.N.A.L.O.G.* magazine number 11)

*A.N.A.L.O.G. IS))is published by A.N.A.L.O.G. Magazine Corp.

LEGAL PROTECTION

No treatise on SOFTWARE PROTECTION would be complete without some discussion of the the LEGAL ways an author can protect his programs. The following is an attempt to give a few highlights in this complicated and vague area. It is not intended to be complete in any sense of the word, but merely an introduction to some of the ways in which you can legally protect your software.

1) Copyrights

A copyright gives a program author the right to prevent others from copying his program once it has been published with an appropriate copyright notice displayed. The Computer Software Copyright Act of 1980 specifically expanded the 1976 Federal Copyright Act to include computer programs. If your program was created after Jan 1,1978, statutory copyright will exist from the date of creation. In other words, if you follow the notice of copyright rules, your right (of copyright) normally applies even though a formal copyright application has not been submitted to the Federal Copyright Office. However, it is a fairly simple and inexpensive process to do the copyright paperwork, and would substantially increase your chances should you ever be required to defend your copyright in court. The copyright method of legal protection for your software should be strongly considered as possibly the best, and the simplest means of legally protecting your software.

Always conspicuously place copyright notices (For example, Copyright (c) 1984, Author Name) on the first page of any documentation, among the beginning lines of the program code, on the outside labels of disks and tapes, on the outside of packages, and any other place that seems appropriate.

2) Trade secret agreements

If you can establish that your program is a trade secret, you may have another way to legally protect it from copying, or unauthorized use by others. To be able to protect your program as a trade secret, it should contain innovative new methods, formulas, patterns, or special compilations of information that give it an edge over other programs, and those innovations must be a secret.

To establish a trade secret type of legal case against a person who has copied or illegally used your program, you must show that the program is is a trade secret, and that the person who used or copied the secret did so through a breach of contract or some special relationship such as an employer/employee relationship, etc.

If you should decide to protect your program in this way, you would not be selling your program to a potential user, but merely licensing him to use it under a confidential software contract. Trade Secret Agreements are normally established by having potential users of your software sign a licensing agreeement to the effect that they may use your software in certain ways, but may not sell, distribute, or allow others to copy or use that software. Trade secret agreements may also be made with employees, who as a condition of employment, agree that they will never divulge or use company secrets for personal gain.

Methods or techniques that are generally known or are public knowledge in the computer industry, cannot be protected as trade secrets. Because public knowledge of a trade secret tends to invalidate it as a secret, trade secrets must be scrupulously guarded and protected against indiscriminate

3) Patents

It is currently a debatable question whether or not computer programs may be patented. In some cases, the courts have upheld patents on computer programs where the program was a part of a total industrial process. However, the use of patents for computer software is such an undefined and nebuloud area that patents are probably not currently the best method of legally protecting your programs. Of course, this situation could drastically change in the future, who knows???

You really should consult an attorney if you decide to follow through on any of these avenues of protection.

One added point to consider regarding legal protection is that it might be a good idea to include your name, initials, and/or a serial number for each program copy, encrypted in secret places inside your program. This technique would provide some strong evidence, should you ever need to prove in court that a particular diskette in question is indeed a copy of your program.
(See Antic * May, 1983 Page 28, article on the INVISIBLE SECTOR.

* ANTIC is Published by ANTIC PUBLISHING, 600 18th Street, San Francisco, CA 94107.

CONCLUSION

Well, now you have quite an aresnal of program protection techniques for the ATARI at your command.

Again, we want to emphasize that there exists considerable controversy regarding protection schemes vs ease of use and backup provision for programs. Obviously, there are valid reasons for copy protecting your programs, but whether or not you will want to use certain protection methods and in particular, those described in this document should depend upon your OWN best judgement and inclinations after weighing all of the pros and cons. It is possible tat making a program difficult to modify or copy may irritate some users who may need to copy for backup or modify for what they feel are legitimate reasons (such as customizing a program for a particular local purpose.)

If you decide to implement at least some of these program protection techniques, then theoretically, the customer will be prevented from making backup copies of your program. In this case, it seems only right that some provision be made for program backup. Merely using double sided diskettes and putting an indntical copy of the program on each side of the diskette would be of some help. In addition, a backup service could be provided whereby a non working original registered diskette would be replaced for a nominal fee.

It is up to you to determine the proper balance for your particular case, and to thoroughly consider the results of your decisions before publication of your protected program.

REFERENCES

The following referenced documents contain information that may be helpful as background information in implementing some of the program protection methods presented in this document:

i) MAGAZINES

A.N.A.L.O.G. COMPUTING, Published bi-monthly by A.N.A.L.O.G. Magazine Corp., 565 Main Street, Cherry Valley, MA 01611. (See magazine number 11, page 72, for article on Disk Backup Programs)

ANTIC - The ATARI Resource, Published twelve times per year by ANTIC Publishine, 524 Second Street, San Francisco, CA 94107. (see May 1983, page 28-article on the INVISIBLE SECTOR)

BYTE, Published monthly by BYTE Publications Inc. 70 Main Street, Peterborough NH 03458. (See Feb, 1982, The Atari Tutorial, Part 6: Atari BASIC)

COMPUTE, The Journal for Progressive Computing, Published 12 times each year by Small System Services, Inc.,P.O.Box 5406, Greensboro, NC 27403 (See april 1983, pages 232-234 for BASIC XIO uses)

CREATIVE COMPUTING, Published monthly by Ahl Computing Inc., a subsidiary of Ziff-Davis Publishing Company,P.O.Box 789-M Morristown, N.J. 07960. (See articles on encryption,

May, 1983 pages 166-195)

InfoWorld, Published weekly by Popular Computing Inc. 375 Cochituate Road, Box 880, Framingham, MA 01701. (See April 18, 1983 Articles on Encryption pages 27 to 35)

2) BOOKS

DE RE ATARI, A Guide to Effective Programming for the 400/800 Home Computer, Published by the ATARI Program Exchange APX-90008

Legal Care For Your Software by Daniel Remer, Addison-Wesley 1982, Reading MA. 01867 (Contains tear out contracts)

3) MANUALS

ATARI 400/800 BASIC REFERENCE MANUAL, Published by the ATARI Software Support Group, P.O.Box 427, Sunnyvale, CA 94086.

ATARI Personal Computer System OPERATING SYSTEM User's Manual, Published by ATARI Inc. Sunnyvale, CA 94086.

4) The following information may also prove helpful:

The address of the United States Copyright Office is

COPYRIGHT OFFICE, LIBRARY OF CONGRESS, Washington D.C. 20559